18:31:43 OCA PAD INITIATION - PROJECT HEADER INFORMATION 05/13/92

Active

Project #: C-36-630 Cost share #: Rev #: 0
Center # : 10/24-6-R7467-0A0 Center shr #: OCA file #:
Work type : RES
Contract#: AGREEMENT DATED 4/14/92 Mod #: Document : AGR
Prime #: Contract entity: GTRC

Subprojects ? : N CFDA:
Main project #: PE #:

Project unit: COMPUTING Unit code: 02.010.300
Project director(s):
SCHWAN K COMPUTING (404)894-2589

Sponsor/division names: NTT NETWORK SYS DEVEL CTR / TOKYO, JAPAN
Sponsor/division codes: 708 / 012

Award period: 920501 to 930228 (performance) 930228 (reports)

Sponsor amount	New this change	Total to date
Contract value	32,000.00	32,000.00
Funded	32,000.00	32,000.00
Cost sharing amount		0.00

Does subcontracting plan apply ?: N

Title: TOWARD VERY HIGH-PERFORMANCE, DEPENDABLE REAL-TIME SYSTEMS

PROJECT ADMINISTRATION DATA

OCA contact: E. Faith Gleason 894-4820

Sponsor technical contact

YUICHI OMACHI, SR. RESEARCH SCIENT.
(146)859-2652

NIPPON TELEGRAPH AND TELEPHONE CORP
NTT NETWORK INFORMATION LABORATORIES
1-2356, TAKE YOKOSUKA-SHI,
KANAGAWA-KEN
238-03 JAPAN

Sponsor issuing office

HIDEKI YOSHIZAWA, SR. MGR. CONTRACTS
(000)000-0000

YOKOSUKA GENERAL AFFAIRS DIVISION
NTT RESEARCH AND DEVELOPMENT HDQTRS.

Security class (U,C,S,TS) : U ONR resident rep. is ACO (Y/N): N
Defense priority rating : N/A N/A supplemental sheet
Equipment title vests with: Sponsor GIT
N/A
Administrative comments -
INITIATION OF PROJECT WITH FOREIGN SPONSOR. CERTAIN EXPORT TRADE REGULATIONS MAY APPLY.

GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT

Closeout Notice Date 01/21/94

Project No. C-36-630 Center No. 10/24-6-R7467-0A0

Project Director SCHWAN K School/Lab COMPUTING

Sponsor NTT NETWORK SYS DEVEL CTR/TOKYO, JAPAN

Contract/Grant No. AGREEMENT DATED 4/14/92 Contract Entity GTRC

Prime Contract No.

Title TOWARD VERY HIGH-PERFORMANCE, DEPENDABLE REAL-TIME SYSTEMS

Effective Completion Date 930228 (Performance) 930228 (Reports)

Closeout Actions Required:	Y/N	Date Submitted
Final Invoice or Copy of Final Invoice	Y	
Final Report of Inventions and/or Subcontracts	Y	
Government Property Inventory & Related Certificate	N	
Classified Material Certificate	N	
Release and Assignment	N	
Other	N	

Comments

Subproject Under Main Project No.

Continues Project No.

Distribution Required:

Project Director	Y
Administrative Network Representative	Y
GTRI Accounting/Grants and Contracts	Y
Procurement/Supply Services	Y
Research Property Managment	Y
Research Security Services	N
Reports Coordinator (OCA)	Y
GTRC	Y
Project File	Y
Other CARL BAXTER-FMD	Y
	N

NOTE: Final Patent Questionnaire sent to PDPI.

Georgia Institute of Technology
College of Computing
Atlanta, Georgia 30332-0280
(404) 894-3152
Fax: (404) 853-9378

Karsten Schwan
schwan@cc.gatech.edu
...!{decvax,hplabs}!gatech!cc!schwan
(404) 894-2589

June 25, 1992

Dr. Yuichi Omachi
Base Systems Architecture Laboratory
NTT Network Information Systems Laboratories
1-2356 Take Yokosuka-shi
Kanagawa-ken, 238-01
Japan

Dear Dr. Omachi:

Thank you for your reminder as to the quarterly report to NTT on the contract entitled "Toward Very High-Performance, Dependable Real-time Systems". I had planned to fax you the report by July 1, but might as well do it now.

Here is what we have done technically:

- **Software models.** We have started working on the report on software models and implementation paradigms by first reviewing research in high-performance parallel operating systems. We are currently writing a review of research in parallel operating systems, of which I should be able to deliver a draft to you by Sept. 1. We are now starting a similar review of the real-time area.
- **OS constructs.** We are now implementing a very high performance light-weight package for object-oriented real-time systems, with the specific goal of portability across multiple parallel machines. We are expecting to have first results of this work available by the end of Summer 1992. In addition, we are developing a high-performance, interactive parallel and real-time program that makes use of this package. The long term goal of this research is to develop a library for parallel and real-time programming that uses object-oriented technology to result in efficient parallel and real-time programs portable across parallel architectures ranging from shared memory to distributed memory multiprocessors. We expect to achieve portability across multiple types of shared memory machines this summer.
- We have some results of our recent research in online scheduling algorithms, but we have not yet applied this work to the problem of designing and implementing real-time and parallel communication protocols (we are starting that research this summer). I will be glad to keep you posted on that research, if you are interested.
- I am a member of the program committee of the annual 'Real-Time Systems Symposium (RTSS'92)', which is the premier national conference for real-time research. Questions: (1) Would NTT be interested in sponsoring a reception during the symposium, during which information on CTRON would be made available? (2) Is NTT planning to send researchers to this conference (taking place in Dec. 1992 in the U.S.)? I would be glad to suggest such a reception during our program committee meeting on July 10, 1992 in Philadelphia. Furthermore, I would be glad to spend time with NTT visitors to this conference, since I am attending it and helping with the organization of its technical program. In addition, if NTT is interested, I would be glad to try to arrange a discussion session on NTT's goals or needs in real-time systems (e.g., CTRON) in conjunction with the annual 'Real-time Software and Operating Systems Workshop (RTOSS'93)' to be held in May 1993. This workshop is the premiere point of contact between academic and industry researchers in the U.S. in the area of real-time systems. Such a session would require that NTT submit an abstract (the deadline for that is likely to be Jan. 1993) and then send one of its real-time researchers to attend the workshop, give

a presentation, and then participate in the session discussion. Since the Dec. 1992 Real-time System Symposium was designed before the start of my contract with NTT, I was not able to try to suggest similar arrangements for RTSS'92, but I would be glad to suggest that something be arranged for RTOSS'93. Furthermore, if RTSS'92 in Dec. 1992 appears too soon for implementation of my idea regarding an NTT sponsored reception, then perhaps you might want to consider such a reception for RTOSS'93?

Last, we are now starting to spend the money you sent to us in April 1992, since we are only now starting the summer quarter. NTT's money (due to our problems with contract negotiation) did not arrive soon enough this spring in order to spend any of the funds during spring quarter (which ends in June 1992). Instead, we have made arrangements to start spending NTT's funds in Summer 1992. However, we HAVE done the work we promised (as you can see from the report above) and we anticipate no additional delays in project completion due to the slight delay in spending the funds you have made available to us.

Sincerely,

Karsten Schwan
Associate Professor

Message 2/14 From Karsten Schwan Sep 10 '92 at 12:28 pm edt

Return-Path: <schwan@cc.gatech.edu>
Date: Thu, 10 Sep 92 12:28:44 EDT
To: barbara@cc
Subject: second report to NTT

Barbara, my second report to NTT has been mailed by now, but I mailed it directly to NTT...can you please accept the following on-line copy and put it in your files and forward it to Simpkin...who needs it, too? Thanks, Karsten

Dr. Yuichi Omachi \\
Base Systems Architecture Laboratory \\
NTT Network Information Systems Laboratories \\
1-2356 Take Yokosuka-shi \\
Kanagawa-ken, 238-01 \\
Japan

Dear Dr. Omachi:

This fax is serving as a progress report for the Sept. 1 due date in our joint work on the
contract entitled ``Toward Very High-Performance, Dependable Real-time Systems''.

Here is what we have done technically:
Software models.
As per my last note, over the summer we did an extensive review of operating systems for parallel machines. I am including an outline of the current version of the report (the actual report is being concurrently sent via airmail). We are NOW ADDING to this report a separate discussion of real-time operating systems. I purposely delayed this addition since I wanted to be able to include the most recent work to appear in the 1992 real-time systems symposium.

OS constructs.
We are still implementing the high performance light-weight package for real-time, threads-based systems. Over the summer, we made good progress on the portability issues involved with the package's construction.

I am including some recent results of our work on online scheduling algorithms in the airmail package being sent, as well.

I am still pursuing a role for NTT at the next Real-Time Systems Symposium. I have spoken with the program committee for the conference and am now talking with the chair of the IEEE committee on real-time systems to explore possible roles for NTT at the conference. I will be sending you a separate fax proposing several possible roles in the near future. In addition, I will keep you posted on other opportunities for NTT involvement or exposure in other outlets for real-time research in the U.S.

Sincerely,

Karsten Schwan, Assoc. Prof.

Command ('i' to return to index):

C 36-630 3+4

SF Grant Conditions (Article 17, GC-1, and Article 9, FDP-11) require submission of a Final Project eport (NSF Form 98A) to the NSF program officer no later than 90 days alter the expiration of the ward. Final Project Reports for expired awards must be received before new awards can be made NSF Grants Policy Manual Section 677).

elow, or on a separate page attached to this form, provide a summary of the completed projects and technical information. Be ure to include your name and award number on each separate page. See below for more instructions.

PART II - SUMMARY OF COMPLETED PROJECT (for public use)

he summary (about 200 words) must be self-contained and intelligible to a scientifically literate reader. Without restating the roject title, it should begin with a topic sentence stating the project's major thesis. The summary should include, if pertinent the project being described, the following items:

The primary objectives and scope of the project
The techniques or approaches used only to the degree necessary for comprehension
The findings and implications stated as concisely and informatively as possible

PART III - TECHNICAL INFORMATION (for program management use)

List references to publications resulting from this award and briefly describe primary data, samples, physical collections, inventions, software, etc. created or gathered in the course of the research and, if appropriate, how they are being made available to the research community. Provide the NSF Invention Disclosure number for any invention.

I certify to the best of my knowledge (1) the statements herein (excluding scientific hypotheses and scientific opinion) are true and complete, and (2) the text and graphics in this report as well as any accompanying publications or other documents, unless otherwise indicated, are the original work of the signatories or of individuals working under their supervision. I understand that willfully making a false statement or concealing a material fact in this report or any other communication submitted to NSF is a criminal offense (U.S. Code, Title 18, Section 1001).

	1-2-93
Principal Investigator/Project Director Signature	Date

IMPORTANT:
MAILING INSTRUCTIONS
Return this *entire* packet plus all attachments in the envelope attached to the back of this form. Please copy the information from Part I, Block I to the *Attention block* on the envelope.

NSF Form 98A (Rev. 8/92)

Georgia Tech

Georgia Institute of Technology
College of Computing
Atlanta, Georgia 30332-0280
(404) 894-3152
Fax: (404) 853-9378

Karsten Schwan
schwan@cc.gatech.edu
...!{decvax,hplabs}!gatech!cc!schwan
(404) 894-2589

March 1, 1993

Dr. Yuichi Omachi
Base Systems Architecture Laboratory
NTT Network Information Systems Laboratories
1-2356 Take Yokosuka-shi
Kanagawa-ken, 238-01
Japan

Dear Dr. Omachi:

This airmail package contains my final report on our joint project entitled "Toward Very High-Performance, Dependable Real-time Systems", to be completed by the end of Feb. 1993. I have instructed Georgia Tech to issue the final invoice for our current project, along with sending you another copy of this final report.

However, let me first say that I much enjoyed meeting you, Dr. Wasano, and Dr. Mori during your visit here Feb. 26. Thank you for your kind gifts and for the delicious dinner! Furthermore, I was pleased to see that your group is obviously well-informed regarding real-time systems research. I am hoping that our interactions during the last year have contributed to this fact.

Later this week, I will also mail two additional items: (1) a brief proposal regarding RTSS sponsorship and (2) a draft of a proposal regarding our future joint research.

I hope that the remainder of your trip to the U.S. was successful and enjoyable, and I trust that you will contact me if you have any further questions regarding the final report on our current project and the new proposals I will be sending you presently.

Sincerely,

Karsten Schwan
Associate Professor

Topics in Real-time Operating Systems: From Applications to Operating System Kernels Final Report to NTT Corporation

Karsten Schwan

Associate Professor
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
February 1993

1 Introduction

This report consists of two parts. In the first part appearing below, we present our own research results and comment on general developments in real-time systems. In the second part, appearing as an addendum, we survey current research in real-time systems and real-time operating systems.

2 Current Results

Our research group is addressing issues in real-time systems that specifically focus on the dynamic aspects of such systems. Namely, we are concerned with real-time system technology that addresses large-scale complex systems that cannot be fully evaluated and timed prior to operation. For example, in large-scale telephone systems, developers typically cannot anticipate all possible failure modes, which suggests that one approach to rapid failure diagnosis and handling is the concurrent execution of telephone network simulations with on-line monitoring and diagnosis software, where simulations are then used to help developers diagnose current or potential problems. Furthermore, in large-scale defense systems (e.g., theater battle management), simple assumptions of worst case event arrivals do not lead to feasible solutions for the on-line software for sensor processing and system control. This is also the case with intelligent autonomous or guided vehicles, where potential overload combinations of external events force software developers to build real-time software that is at least equally concerned with how to react to and process unexpected events than with the standard processing of low-level events. Moreover, in some intelligent autonomous systems, such problems are exacerbated by the presence of severe restrictions on packaging that limit the processing power available onboard the vehicles (e.g., the NASA Mars Rover).

Real-time Operating System Kernels. The facts listed above have prompted our research group to pursue research specifically addressing dynamic events in real-time systems. In the Chaos series of systems, we have produced a succession of operating system kernels for multiprocessors – termed the CHAOS [14, 15, 5, 6] and GEM systems for implementation of real-time program and on-line program adaptation[1, 2]. The purpose of these systems was to evaluate the feasibility of an alternative approach to making real-time guarantees. Namely, rather than attempting to evaluate a program's timing properties prior to program execution and then enforcing such properties at runtime, we assume that runtime timing errors will occur and therefore, also offer additional operating and programming system mechanisms that permit programmers to evaluate the actual timing properties of program components during program execution and then execute alternative actions – termed *reactive adaptations* – when timing errors are detected. In addition, we offer mechanisms that permit programmers to anticipate timing errors and attempt to take corrective actions before timing errors actually occur in the running system – termed *preventive adaptations.* Since such programming requires that real-time systems can deal with unanticipated external events, giving rise to new tasks for execution and the deletion of existing tasks, we must also solve

the problem of on-line scheduling for time-constrained tasks. The Chaos system is currently available on SUN Sparcstations and is now being ported to a Kendall Square multiprocessor supercomputer.

On-line Scheduling. One of the premier issues addressed by our group during the last year has been the on-line scheduling of time-constrained tasks. Zhou addressed these issues for tasks with hard execution deadlines, for uniprocessor and multiprocessor real-time systems[3, 16, 17]. This has resulted in the development of a real-time threads package on the BBN Butterfly multiprocessor, which is now being ported to more modern machines, specifically a 32-node Kendall Square Research supercomputer. The uniprocessor and multiprocessor scheduling performed in this package has demonstrated the performance limitations of efficient on-line hard deadline scheduling. These results are now causing us to re-design our low-level implementation of real-time threads such that even at the threads level, existing mechanisms are easily changed – termed configurable multiprocessor threads[12]. Our current results concern the on-line and off-line adaptation of thread locking constructs (mutex locks) and the adaptation and exchange of schedulers embedded in the threads package. We are also extending our results regarding multiprocessor scheduling to the scheduling required for autonomous vehicle applications. The specific work being performed addresses both schema-based vehicle navigation and the scheduling of primary and alternate actions for navigation in guided systems. Both topics essentially address the on-line scheduling of groups of real-time tasks[4]. Furthermore, Ghosh is addressing the issue of on-line scheduling for real-time simulations[9, 8, 7], where we are specifically focussing on simulations written using optimistic methods like TimeWarp. Last, we are beginning to look at reducing the cost of on-line scheduling by use of timing (or deadline) semantics that are specific to certain applications rather than generic to all real-time programs. For example, in multi-media applications on computer networks, it is not necessary to guarantee the deadlines of each frame being sent across the net. Instead, vaguer formulations like 'ensuring timely progress' may be more suitable, and they are likely to give rise to more efficient runtime scheduling algorithms.

On-line Program Adaptation. Real-time systems cannot be made adaptive by simply adding on-line scheduling algorithms to those system. Instead, we must also develop mechanisms for the detection of and reaction to unexpected events. This is the purpose of the Falcon system for on-line program monitoring and evaluation of monitoring information now being constructed by our group. The purpose of the Falcon system is to provide a basis for the construction of adaptive real-time systems, again building on the configurable threads library employed by our group. With Falcon, we are performing experiments with the adaptation of scheduling for TimeWarp simulations (the selective pre-execution of computations when expecting increases in event arrivals), with the on-line adaptation of certain program components (e.g., mutex locks in configurable threads), and most importantly, with on-line program steering. Namely, we are interested in future systems that will permit program implementors, end users, and on-line algorithms to interact with parallel and real-time applications during program execution. Such interactions have been identified as 'program adaptations' by the real-time community[2] and as 'interactive program steering' by the scientific community.

The Falcon system addresses interactive program adaptation or steering on parallel and distributed machines, ranging from small-scale shared memory machines like Silicon Graphics multiprocessors to large-scale machines like the Kendall Square Research multiprocessor and the Intel Paragorn, to sets of networked workstations.

3 New Directions 1: Monitoring for Reactive Systems

We are interested in future systems that will permit program implementors, end users, and on-line algorithms to interact with parallel and real-time applications during program execution. The Falcon system addresses several components of the interactive and real-time program adaptation or steering tasks. Specifically, the 'Falcon' system is useful for the on-line monitoring of threads-based programs on parallel and distributed machines, ranging from small-scale shared memory machines like Silicon Graphics multiprocessors to large-scale machines like the Kendall Square Research multiprocessor and the Intel Paragorn, to sets of networked workstations. The system has several essential capabilities:

- Via an easy-to-use graphical interface, it can perform default monitoring of arbitrary threads programs, where users may select from multiple 'monitoring views' that differ in monitoring perturbation vs. the amounts of provided program information.
- Monitoring overheads are kept small by use of both the built-in monitoring hardware offered by parallel machines and of code fragments called sensors and probes included with user code. An easy-to-use system for inclusion of sensors with user code is also part of the Falcon system. In addition, we are now investigating the attachment of real-time constraints to Falcon monitoring, which is quite important for future large-scale real-time systems.
- A major difference of Falcon from other monitoring tools like Paragraph[10] is the system's ability to accept user-specifications that state what and how monitoring should be performed for a specific program. As a result, users can easily construct just the monitoring views they desire, in addition to or instead of the default views provided with the system. This capability of Falcon is essential for the construction of program adaptation algorithms that use application-level knowledge by users in order to improve runtime peformance of real-time programs.
- Falcon permits the attachment of either on-line adaptation algorithms or graphical views to monitoring views[13, 11]. This allows the construction of systems that can both react to program changes detected by monitoring in real time and record the changes and reactions for postmortem analysis and display.

4 New Directions 2: Real-time Intelligent Autonomous Systems

A major problem in autonomous systems is the difference in temporal assumptions and requirements made at the knowledge-based vs. control levels of such systems. For low-level controls, systems must abide by the temporal constraints imposed by the physical world in which they operate or risk failure. For example, joints must be rotated at speeds within their structural specifications or within power availability. Furthermore, at the control level, potential catastrophic failures give rise to hard or at minimum, soft real-time requirements, while many computational techniques employed in knowledge-based systems exhibit large temporal variances. The goal of our research is not to limit the temporal variance in knowledge-based systems. Instead, we wish to use temporally variant tasks in real-time systems such that we can still provide the hard or soft temporal guarantees sought by lower level real-time software. We view this dichotomy between hard real-time and knowledge-based sub-systems as being central to the problem of complex system control.

Our approach to addressing the temporal variance of knowledge-based systems is to use additional processing capacity, thereby capitalizing on current trends regarding the availability of relatively cheap multiprocessor systems. Specifically, the topic of our research is the development of operating system mechanisms that facilitate the use of tens to hundreds of cooperating processors connected in a single autonomous system. These mechanisms will address two specific issues:

1. Providing temporal guarantees for tasks with large variances in execution time.
2. Providing high performance data sharing for distributed tasks via by-value consistency.

5 New Directions 3: Time-Constrained Optimistic Simulation

Recent research and development in real-time applications is resulting in the construction of increasingly complex, large-scale, real-time systems. Such systems are complex for two reasons: (1) the hardware and software components of such systems are extensive and (2) the correct operation of such systems requires that they execute under well-defined timing constraints in execution environments that generate and demand reactions to unexpected tasks. These may include temporary overloads, and on-line reconfiguration or adaptation of selected system components.

The goal of this research is to assist the developers of complex real-time systems, by permitting them to execute fully implemented system components in conjunction with simulations of the remaining system. Such executions will honor the timing constraints of the final system on both the simulated and completed system components. As a result, a simulated

system component must have the ability to emulate its final implementation:
(1) the simulated component must execute within well-defined timing constraints,
(2) its execution may be periodic, with its period corresponding to that of its implemented counterpart(s), or
(3) its execution may be sporadic, in response to dynamically arriving sporadic tasks. Since we are using optimistic simulation methods (due to their proven advantages regarding execution speed), the simulator may have to recover from incorrect executions of events using rollback. This presents us with a scheduling problem where both future events and recovery actions for events must be scheduled such that deadlines are not missed. We note that this problem, when solved in Timewarp simulations, will also contribute to the general literature on dynamic systems, where events may often be anticipated, pre-executed, and where recovery has to be performed as well, either as standard or as forward recovery. Our work employs a window-based throttling algorithm to control the amount of pre-execution, where window length is controlled dynamically and adaptively.

6 Conclusions

With the NTT funding, we have performed some of the research explained above. In addition, we have worked on increasing the profile of NTT at real-time system symposia in the U.S. Last, we have surveyed current research results in real-time operating systems and this survey will be made available to NTT by the end of February 1993 as part of the official final report for this project.

We expect to continue our cooperation with NTT, but would like NTT to consider sending a researcher to Georgia Tech for some time, thereby improving our collaboration. Such an exchange would also allow us to introduce NTT researchers to the significant research being performed at Georgia Tech in the engineering domains of manufacturing, in the areas of remote learning and telecommunications, etc.

References

[1] T. Bihari and K. Schwan. A comparison of four adaptation algorithms for increasing the reliability of real-time software. In *Ninth Real-Time Systems Symposium, Huntsville, AL*, pages 232–241. IEEE, Dec. 1988.

[2] T. Bihari and K. Schwan. Dynamic adaptation of real-time software. *ACM Transactions on Computer Systems*, 9(2):143–174, May 1991. Older version available from the Department of Computer and Information Science, The Ohio State University, OSU-CISRC-5/88-TR, newer version available from College of Computing, Georgia Institute of Technology, Atlanta GA, GTRC-TR-90/67.

[3] Ben Blake and Karsten Schwan. Experimental evaluation of a real-time scheduler for a multiprocessor system. *IEEE Transactions on Software Engineering*, 17(1):34–44, Jan. 1991.

[4] Harold Forbes and Karsten Schwan. Dynamic scheduling in knowledge-based real-time applications. In *NSF Workshop on Artificial Intelligence in Real-Time: AI Support for Mission-Critical Computing Applications, Univ. of Maryland, College Park.* NSF and DARPA, Feb. 1993.

[5] Ahmed Gheith and Karsten Schwan. Chaos-art: Kernel support for atomic transactions in real-time applications. In *Nineteenth International Symposium on Fault-Tolerant Computing, Chicago, ILL*, pages 462–469, June 1989. Also see GIT-ICS-90/06, College of Computing, Georgia Tech, Atlanta, GA 30332.

[6] Ahmed Gheith and Karsten Schwan. Chaos-arc – kernel support for multi-weight objects, invocations, and atomicity in real-time applications. Technical report, GIT-ICS-90/06, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, Jan. 1990. To appear in ACM TOCS.

[7] Kaushik Ghosh, Richard M. Fujimoto, and Karsten Schwan. Speculative execution in time-constrained systems. Technical Report GIT-CC-92/22, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, May 1992.

[8] Kaushik Ghosh, Richard M. Fujimoto, and Karsten Schwan. A testbed for optimistic execution of real-time simulations. *Proceedings of the IEEE Workshop on Parallel and Distributed Real-Time Systems*, April 1993.

[9] Kaushik Ghosh, Richard M. Fujimoto, and Karsten Schwan. Time warp simulation in time constrained systems. *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS)*, May 1993.

[10] Michael T. Heath and Jennifer A. Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, 8(5):29–39, Sept. 1991.

[11] Carol Kilpatrick and Karsten Schwan. Chaosmon – application-specific monitoring and display of performance information for parallel and distributed systems. In *ACM Workshop on Parallel and Distributed Debugging*, pages 57–67. ACM SIGPLAN Notices, Vol. 26, No. 12, May 1991.

[12] Bodhisattwa Mukherjee. A portable and reconfigurable threads package. In *Proceedings of Sun User Group Technical Conference*, pages 101–112, June 1991.

[13] David M. Ogle, Karsten Schwan, and Richard Snodgrass. The dynamic monitoring of real-time distributed and parallel systems. Technical report, College of Computing, Georgia Institute of Technology, ICS-GIT-90/23, Atlanta, GA 30332, May 1990. To appear in IEEE TSE.

[14] Karsten Schwan, Win Bo, and Prabha Gopinath. A high-performance, object-based operating system for real-time, robotics applications. In *1986 Real-Time Systems Symposium, New Orleans, Louisiana*, pages 147–156. IEEE, Dec. 1986.

[15] Karsten Schwan, Prabha Gopinath, and Win Bo. Chaos – kernel support for objects in the real-time domain. *IEEE Transactions on Computers*, C-36(8):904–916, July 1987.

[16] Hongyi Zhou. *Task Scheduling and Synchronization for Multiprocessor Real-Time Systems*. PhD thesis, College of Computing, Georgia Institute of Technology, May 1992.

[17] Hongyi Zhou, Karsten Schwan, and Ian Akyildiz. Performance effects of information sharing in a distributed multiprocessor real-time scheduler. Technical report, College of Computing, Georgia Tech, GIT-CC-91/40, Sept. 1991. Abbreviated version in 1992 IEEE Real-Time Systems Symposium, Phoenix.

Survey of Real-Time Operating Systems

Bodhisattwa Mukherjee (bodhi@cc.gatech.edu)
Karsten Schwan (schwan@cc.gatech.edu)

GIT–CC–93/18

March 1, 1993

Abstract

This paper describes current research in real time operating systems. We first present recent results in real-time task scheduling. Next, we summarize research in real-time synchronization, followed by a discussion of the structures and primitives offered by selected real-time operating system kernels.

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332–0280

1 Introduction

The embedded computer hardware of modern robots and industrial control systems is becoming increasingly complex. Typically, it consists of many interconnected computers operating at multiple levels of control or supervising different mechanical or electronic system peripherals. Given such hardware, the efficient execution of a real-time application requires that programmers deal with issues that arise for other high-performance, parallel and distributed application programs, such as efficient resource management, task and communication scheduling, load balancing, and programmed dynamic reconfiguration and recovery. Therefore, as with parallel and distributed operating systems, a real-time operating system must provide programmers with primitives for task control, interprocess communication, device operation, etc. However, real-time operating systems must also offer support specific to the real-time domain:

- In contrast to single, large-scale parallel or distributed applications, most complex real-time applications exhibit multiple grains of parallelism. Therefore, the operating system must support parallel application tasks of differing sizes, ranging from small tasks executed at high rates and by necessity consisting of a small number of instructions, to large tasks executed infrequently.
- Since the current and future target architectures of real-time application vary widely (ranging from the small embedded systems in the autonomous Mars Rover to future drill hole sensor processors of the power of a CM-5), the corresponding real-time operating system kernels must vary in size and functionality, as well. Therefore, real-time kernels must be highly configurable in size and functionality.
- Tasks are time-critical, where task deadlines may vary in semantics and in laxity, including indications of task periodicity, recoverability, criticality for periodic tasks, etc. As a result, while tasks must be executed within application-specific timing constraints, no single task scheduler is likely to satisfy all real-time applications. This constitutes another reason for operating system configurability, even at the lowest operating system levels.
- Time constraints in task execution imply time constraints in task communication, as well. Furthermore, a single model of task communication (e.g., RPC) has been shown unsuitable for many real-time applications. Specifically, tasks may make different assumptions regarding the model of communication used. For example, some tasks may tolerate the loss of individual readings from a sensor in order to perform an operation asynchronously at the highest rate possible, whereas other tasks may assume that individual messages never get lost. As a result, real-time operating systems should support multiple models of time-critical task communication.

This paper surveys recent research in the area of real-time operating system. The survey focusses on the novel research performed during the last few years, rather than commercially available systems like pSOS, CTRON, etc. In the remainder of this paper, we first describe

some results in real-time task scheduling, because this area is of critical importance to real-time operating systems. Section 3 elaborates on synchronization in real-time systems. Section 4 describes the salient features of experimental real-time operating system constructed during the last few years.

2 Real-Time Scheduling

Research on real-time scheduling has experienced a major shift during the last few years, from static to dynamic (or on-line) real-time scheduling. We begin with a brief review of static scheduling, followed by a discussion of dynamic scheduling. For a more extensive review of research in static real-time scheduling, the reader may refer to [CSR88]. All work presented above uses the same measures of 'success' or 'quality' for real-time scheduling algorithms: algorithm complexity (worst case running time) and the algorithms' ability to meet the deadlines of a given set of tasks to be scheduled (in the static case) or to perform as well regarding the deadlines being met as any other available algorithm (in the dynamic case).

2.1 Well-known Scheduling Algorithms

Algorithms. Early research focusses on relatively small-scale or static real-time systems, where task execution times can be estimated prior to task execution (i.e., data dependies are limited), and where the resulting task schedules can be determined off-line. The resulting scheduling algorithms address both periodic tasks and sporadic tasks, where periodic tasks typically arise from sensor data and control loops, while sporadic tasks arise from asynchronous events or operator actions. A scheduling algorithm jointly schedules the periodic and the sporadic tasks so that the timing requirements (such as deadlines, execution rates, etc.) for both set of tasks are met.

Most popular among static algorithms are the **Rate Monotonic Algorithms (RM)**, in part because they are easily mapped to priority-based low-level task schedulers. The basic idea of the rate monotonic algorithm is to assign fixed priorities to tasks with different execution rates, highest priority being assigned to the highest frequency tasks, lowest priority to the lowest frequency task. At any time, the low-level scheduler simply chooses to execute the highest priority task. By specifying the period and maximum computation time of each task, the behavior of the system can be categorized a priori[SLR86].

One problem with RM algorithms is their lack of support for dynamically changing periods, which is being addressed in several publications from that community of researchers[SSL89,

BSS88]. Furthermore, the schedulable bound[1] is less than 100%. In [HKL91], the authors consider the problem of fixed priority scheduling of periodic tasks where each task's execution priority may vary. A method for determining the schedulability of each task is presented.

A second problem with RM scheduling is priority inversion, which is the situation where a high priority job must wait for a lower priority job to execute. In [MT92], the authors consider the nature of the non-preemptable critical regions which give rise to such priority inversion in the context of a soft real-time operating system, where average response time for different priority classes is the primary performance metric. An analytical model is described which is used to illustrate how non-preemptable critical regions may affect the time-constrained jobs in a multi-media (soft real-time) task set. Chen et. al.[CL91] studies a priority ceiling protocol for multiple-instance resources. They present an optimal resource-allocation algorithm which can be used to improve the schedulability of a real-time system.

In contrast, **Earliest Deadline First (EDF) Scheduling Algorithms** can be used for dynamic as well as static scheduling[ZRS87b, DM89, SZ92]. This algorithm uses the deadline of a task as its priority. The task with the earliest deadline has the highest priority. Since priorities are dynamic, the periods of tasks can be changed at any time. A variant of EDF scheduling is **Minimum-Laxity-First Scheduling (MLF)**, where a laxity is assigned to each task in the system, and minimum laxity tasks are executed first. Laxity measures the amount of time remaining before a task's deadline will pass if the task uses its alloted maximum execution time. Essentially, laxity is a measure of the flexibility available for scheduling a task. The main difference between MLF and EDF is that unlike EDF, MLF takes into consideration the execution time of a task[SK91].

While EDF is superior to RM in the sense that its schedulable bound is 100% for all task sets, a problem with EDF is that there is no way to guarantee which tasks will fail in transient overload situations. This has resulted in another variant of EDF scheduling, called the **Maximum-Urgency-First Algorithm (MUF)**[SK91], where each task is given an explicit description of urgency. This urgency is defined as a combination of two fixed priorities, and a dynamic priority, which is inversely proportional to the task's laxity. One of the fixed priorities, called task criticality, has precedence over the task's dynamic priority. The other fixed priority, called user priority, has lower precedence than the task's dynamic priority. The idea is to use user-specified notions of 'priority' to help on-line algorithms distinguish more important from less important tasks.

In [Jef92] an optimal algorithm is presented for scheduling a set of sporadic tasks that share

[1]The schedulable bound of a task set is defined as the maximum CPU utilization for which the set of tasks can be guaranteed to meet their deadlines.

a set of serially reusable, single unit resources such that tasks complete executions before a deadline and resources are accessed sequentially. The algorithm combines EDF scheduling with a synchronization scheme for access to shared resources. Synchronization is also addressed in [SZG91, ZSG92].

Evaluation and Improvements. Researchers have studied both preemptive[LL73, MD78, LM80, Law81, Mok83, RS84, SLR86, ZS91] and non-preemptive [Lei80, LY82, ZRS87a] scheduling algorithms extensively in the context of real-time systems. In [LL73], The authors showed that the rate-monotonic and earliest deadline scheduling algorithms are optimal static priority and dynamic priority scheduling algorithms respectively in a uni-processor preemptive scheduling environment. In [Mok83], the author proved that the slack-time algorithm is optimal too.

In [MD78], the authors have defined an optimal run time scheduler for the hard real-time environment[2] as one which is able to meet all task dead lines, provided that one exists. They showed that both earliest due date (EDD) and least laxity first (LLF) sequencing of arrivals qualify in this definition as optimal run-time schedulers. They also showed that an optimal scheduler cannot be found for multiprocessors unless a priori knowledge exists of the deadlines, computation times and arrival times of all the tasks. In [GJ77], the authors showed that even for a single processor, constructing a schedule with arbitrary arrival, computation and laxity requirements is NP-complete.

The performance study in [JLT85] of various classical scheduling algorithms also considers situations where computation times are not exactly known at the time of task arrival, but have some given, known distributions. The study then introduces the notion of *value function* that specifies the value of completing a task at any time after arrival. This unifies the treatment of both hard and soft real-time environments; tasks with value functions that became negative while waiting or during execution are discarded from the queue or aborted. Since the task execution times were not exactly known at the initiation time of a task, it was possible to have discarded/wasted work. The best algorithm in terms of maximizing the value of completed tasks for multiprocessor are a pair of heuristic schedulers that take into account the expected value of a task at completion. The expected value of a task is the probability that a task completes prior to its critical time or deadline. This study does not address how suitable values functions may be found for practical scheduling problems.

Recent work in real-time scheduling addresses uniprocessor and multiprocessor systems, where scheduling must be performed on-line for both sporadic and periodic arrivals. One of the few on-line multiprocessor algorithms in the literature is an any fit algorithm proposed

[2] In a hard real-time environment, a task which expires while in the queue is discarded and not considered for service, whereas in a soft real time environment, such a task is retained in the queue and is still eligible for service.

by Blake and Schwan[BS91b], which offers a distributed implementation consisting of global schedulers performing the assignment of tasks to processors in cooperation with processor-local schedulers that carry out deadline scheduling. This algorithm is similar to algorithms developed for distributed systems, such as bidding algorithms [RS84]. This work is extended further in [ZSA91].

In [KS92], the authors present an optimal on-line scheduling algorithm for overloaded systems. In [KM92], the semantics of data-intensive real-time applications are discussed. By examining the semantics of these applications, the concept of similarity is formed, which has been used on an ad hoc basis by application engineers to provide more flexibility in concurrency control. An efficient real-time scheduling algorithms exploiting similarity is proposed.

Effects of Cycle-Stealing on Scheduling Algorithms. In [RSL87], the authors discuss the effects of cycle stealing on scheduling algorithms in a hard real-time environment. An I/O device can transfer data by direct memory access (DMA) and steal cycles from the processor and therefore from the executing task. In a real-time systems, cycle-stealing can cause delays which lead to the missing of task dead-lines at a low degree of processor utilization. Often, I/O devices are designed in such a way that FIFO is the only possible way to schedule I/O activity. The benefits gained by an intelligent algorithm to schedule the processor can be negated by such an I/O scheduling algorithm[SLR86]. In [RSL87], the authors address the issues raised by the integration of the scheduling of processor and I/O devices. They propose remedies to the problem in two steps – first, the I/O devices should be more intelligent so that they can be scheduled just as the CPU. Secondly, they determine the degree of inter-leaved memory needed to effectively counter the effects of cycle stealing. When both I/O in the DMA controller and computation tasks in the processor are schedulable, the provision of only 4 banks of low-bit interleaved memory can lead to significant improvement in performance. With 8 banks near-optimal performance is obtained.

Imprecise Computations. Recent work in real-time scheduling is also considering changes in the semantics of timing constraints to be used and enforced in actual systems. This is most visible in recent research on multi-media systems, and it is also seen in the more theoretical problem formulations advanced by the 'imprecise computation' community and more recently, by work in 'anytime' algorithms being performed by AI researchers. The basic idea of that work is that there are some algorithms (e.g., iterative algorithms) that can return results at almost any time during their execution. The longer they run, the more precise their results. Ideally, a process executes until a result with a desirably small tolerance has been obtained. When time is

limited, one can terminate the process before it completes a sufficient number of iterations. The result produced by a prematurely terminated process may be not as precise as desired, but it may still be acceptable, and therefore, can be used by the application. In [LLN87], the authors discuss a formulation of this problem taking into the account the quality of the overall result. A few scheduling algorithms for such computations are also presented in the paper. In [SL92], the authors describe three algorithms for scheduling preemptive, imprecise tasks on a processor to minimize the total error. Each imprecise task consists of a mandatory task followed by an optimal task. Some of the tasks are on-line; they arrive after the processor begins execution. The algorithms assume that when each new on-line task arrives, its mandatory task and the portions of all the mandatory tasks yet to be completed at the time can be feasibly scheduled to be completed by their deadlines. The algorithms produce for such tasks feasible schedules whose total errors are as small as possible. Three algorithms are presented for three types of task systems: (1) when every task is on-line and is ready upon its arrival, (2) where every on-line task is ready upon arrival but there are also off-line tasks with arbitrary ready times, and (3) when on-line tasks have arbitrary ready times.

Recent research in anytime algorithms does not assume mandatory task components, and practical research in robotics applications generally does not use imprecise algorithms. Instead, alternative sets of algorithms are employed[BS91a].

Real-time Distributed Systems. Most distributed scheduling algorithms have two common features[WC87]: (1) a global task sharing strategy between nodes and (2) a local scheduling policy for individual node. The local scheduling policy is often based on heuristics that find which task to accept and which task to reject in an efficient manner. In [CL86], the authors compare a few distributed scheduling algorithms based on the means utilized to share information (i.e., the global part). In [RS84], the authors propose a heuristic for local scheduling in a distributed system called the "guarantee" routine. Here, an arriving task is inserted into the queue if it is possible to guarantee that both the arriving task and all other tasks currently in the queue do not miss their deadlines. Rejected tasks are then passed on to the task sharing algorithm for distribution to other processors.

Local scheduling algorithm performance for hard real-time distributed systems is determined not only by the rejection ratio but the number of tasks able to be shared. This in turn depends on the rejected task having sufficient laxity remaining at rejection to enable it to be sent to other processors. For tasks with hard real-time deadlines, results are limited to FCFS service. The earliest work in this area was presented by Gnedenko and Kovalenko in [GK68]. They present

analytic results for the ratio of rejected tasks arriving to a multiprocessor system with Poisson arrivals and tasks with known exponential computation and laxity requirements. In [BBH84], the authors present transform solution results for tasks with arbitrary computation, laxity and arrival distributions where FCFS service is used and all task parameters are known on arrival to the queue. The authors in [KSC86] extend analytic results for an elementary load sharing algorithm with tasks of fixed laxity and a fixed delay in node sharing, again assuming a FCFS local scheduling algorithm. In [WC87], the commonly used local non-preemptive algorithms are examined, and their performance is compared with regard to rejection ratio and expected task laxity at rejection. The policies compared are the standard sequencing methods of FCFS, SJF (sequencing by shortest computation time first), LLF, EDD, the local "guarantee" routine (GM), and a run-time selection algorithm (called MM algorithm) based on the Moore ordering rule[Moo68]. The criteria considered for selection of a local scheduling algorithm for hard real-time systems is that of minimum rejection ratio, maximum number of rejected tasks with positive laxity and greatest task laxity at rejection for tasks with positive laxity. The simulation shows that MM algorithms give the best performance for task rejection for a given example of task computation and laxity. For the other criteria, the FCFS algorithm gives the largest number of rejected tasks with positive laxity while the LLF algorithm gives the greatest laxity at rejection. The CPU utilization appears to be similar for the LLF, EDD, GM and MM algorithms and for the FCFS and SJF algorithms.

In [HS92], the authors address the problem of allocating (assigning and scheduling) periodic task modules to processing nodes (PNs) in distributed real-time systems subject to task precedence and timing constraints. They propose a module allocation algorithm (MAA) to find an "optimal" allocation that maximizes the probability of meeting task deadlines using the branch-and-bound technique. The task system within a planning cycle is first modeled with a task flow graph (TG) which describes computation and communication modules as well as the precedence constraints among them. To incorporate both timing and logical correctness into module allocation, the probability of meeting task deadlines is used as the objective function. The MAA is then applied to find an optimal allocation of task modules in a distributed system. The timing aspects embedded in the objective function drive the MAA not only to assign task modules to PNs, but also to use a module scheduling algorithm (MSA) (with polynomial time complexity) for scheduling all modules assigned to each PN so that all tasks may be completed in time. Several numerical examples are presented to demonstrate the effectiveness and practicality of the proposed algorithms.

In [SZG91], the authors address an important problem regarding a large distributed memory multiprocessor which is best stated by the following two questions: (1) how are the latency and the quality of scheduling affected by different degrees of completeness in the information shared among multiple, potentially concurrent schedulers? and (2) how can scheduling information be represented so that it is efficiently and concurrently accessible? The authors have dealt with these problems for one important class of parallel programs: real-time applications executing on dedicated multiprocessors. Specifically, they present a real-time scheduling algorithm for multiprocessors that is scalable in the number of tasks performing scheduling and in the maximum amount of computation time consumed by those tasks. They also develop a flexible representation for shared information within the distributed scheduler that is easily varied regarding its degree of information completeness. They then show that the sharing of incomplete (vs. complete) information can lead to increased performance regarding scheduling latency with few or no losses in scheduling quality.

In [HS91], Hou et. al. propose a load sharing algorithm for real-time applications which takes into account the effect of future task arrivals on locating the best receiver for each unguaranteed task in a heterogeneous distributed environment.

2.2 Future Work

Current and future work in real-time scheduling must address the highly dynamic, complex environments of large-scale real-time systems, such as national networks carrying time-constrained communications (e.g., multi-media applications such as collaboration systems), or large-scale theater battle management systems, or large-scale simulations. As a result, researchers are addressing on-line scheduling, scheduling for parallel and distributed systems, and they are addressing the semantics of timing constraints to be enforced in such future systems (e.g., hard deadlines are simply neither needed for feasible as a formulation of timing constraints in multi-media applications). Researchers are also addressing the actual overheads experienced by scheduling algorithms. This is resulting in more attention paid to scheduling algorithm and scheduler implementation rather than considerations of algorithm optimality and complexity (the latter only captures worst case performance, whereas we are really interested in average case performance in actual systems). In addition, new topics like reliability coupled with timeliness must be explored for large-scale systems.

3 Synchronization

Synchronization in multiprocessor hard real-time systems is a relatively new field. An important problem that arises in the context of such real-time systems is the effect of blocking caused by the need for synchronization among tasks that require exclusive access to shared logical or physical resources. Mok[Mok83] showed that the addition of mutual exclusion requirements in real-time programs makes the general scheduling problem an NP-hard problem.

For uniprocessor systems running periodic tasks, two recent protocols provide effective solutions to the scheduling problem with resource sharing. They are the *kernelized monitor* protocol[Mok83] and the *priority ceiling* protocol[SRL90]. In the kernelized monitor protocol, the *earliest deadline first* scheduling policy is used for task scheduling. All executions in critical sections are nonpreemptable. However, schedulability analysis performed in this protocol requires the use of upper bounds on the execution times of all critical sections appearing in tasks. Since such upper bounds may be overly pessimistic, the use of the kernelized monitor protocol may result in low processor utilization.

The priority ceiling protocol is designed for systems where each task has a fixed priority and the *rate monotonic* scheduling algorithm is used. With this protocol, Sha et al.[SRL90] showed that in the worst case, each task only has to wait for at most one lower priority task to finish in a critical section, and deadlocks cannot occur. Assuming that the longest possible waiting time is known for each task in the system, sufficient conditions for scheduling sets of periodic tasks have also been derived. However, the priority ceiling protocol cannot be directly used when priorities are dynamic. This is addressed by the recent work of M. Chen et al.[CL90] who have extended the original priority ceiling protocol to one which is able to handle dynamic priorities.

For uniprocessor systems, K. Jeffay[Jef89b] has developed schedulability conditions for a set of sporadic tasks that each consist of a sequence of *phases* each of which may require access to at most one shared resource. In his analysis, tasks' timing constraints as well as resource requirements are assumed to be known beforehand. He showed that optimal synchronization and scheduling disciplines exist for restricted patterns of resource usage.

Predictable synchronization on multiprocessor real-time systems offers a new challenge compared to existing work on uniprocessor synchronization and scheduling[Jef89b, SRL90, CL90]. In [MSZ90], Molesky, Shen and Zlokapa have described predictable algorithms for semaphores with linear waiting. Although their proposed algorithms are predictable, they do not take into

account the priorities of the processes that want to acquire the semaphore. In [RSL88], the authors have presented a multiprocessor extension of the priority ceiling protocol [RSL89]. The priority ceiling protocol minimizes priority inversion for a set of periodic real-time processes that access exclusively some shared data. The multiprocessor priority ceiling protocol generalizes the uniprocessor priority ceiling protocol by executing all the critical regions associated with a semaphore on a particular processor called synchronization processor. So, the critical regions in the programs are substituted by an invocation to a remote server that deals with all the critical regions associated with a particular semaphore. The existence of a remote centralized server often limits the scalability of the solution, and increases the cost of executing fine grain sharing applications.

In [ZSG92], the authors study mutual exclusion and synchronization for *dynamic* hard real-time multiprocessor applications. As with any dynamic parallel program, a dynamic real-time application's execution can result in on-line creation of additional tasks, and the creation of such time-constrained tasks cannot be predicted or accounted for prior to program execution. The research results presented in this paper concern task synchronization such that guarantees can be made regarding the synchronized tasks' timing constraints. Such guarantees cannot be made without performing on-line schedulability analysis and on-line analysis concerning the maximum time that a task will wait for some resource being acquired with a synchronization primitive. They present a real-time locking scheme that prevents deadlocks and ensures time-bounded mutual exclusion. The maximum waiting time for a task attempting to acquire a resource is computed with an $O(1)$ algorithm. Two important attributes of the algorithm are: (1) previously made guarantees regarding resource accesses are always maintained, and (2) failures regarding accesses to shared resources are reported immediately. As a result, the application program or higher-level operating system software can deal with such failures in a timely manner, by acquisition of alternative resources, by execution of exception handling code, etc.

Anderson [And90] and Mellor-Crummey and Scott [MCS91] derived spinlock implementations that service lock requests in FIFO order and can be used in real time systems. In [Mar91], the author has defined a synchronization mechanism called the priority spin lock and has suggested algorithms to implement priority spin locks with local spinning. A priority spinlock has a priority ordering property. Each processor that competes for a priority spinlock has a unique dynamic priority that reflects the importance of the process it runs.

3.1 Future Research

No clear direction can be discerned regarding future work in real-time synchronization.

4 Real-Time Kernels and Run Time Systems

4.1 Current Work

Research on real-time operating systems in the U.S. has been driven by three primary concerns: (1) support of the Ada language, (2) the predictable execution of embedded systems (e.g., the GEM, Spring, Arts, and YartOS kernels, and real-time threads and Mach), and (3) dealing with the complexity of large-scale and dynamic real-time applications (e.g., the CHAOS kernels). Furthermore, recent research is becoming more concerned with the provision of some platform on which diverse real-time systems may be constructed (e.g., real-time, configurable threads and micro-kernels). Commercial systems, on the other hand, have typically provided either Ada support (e.g., environments supported by Honeywell or TRW), or some fixed set of primitives (i.e., micro-kernels) at the process level (e.g., pSOS), or they have focussed on building real-time extensions to or alterations of Unix, the latter now resulting in the POSIX real-time standards for Unix (which basically state that the low-level scheduler in Unix shall be priority based and may be exchanged for a different scheduler). The DOD is also supporting real-time Mach, but that project has not yet delivered usable systems to any target installations.

Our research has focussed on dynamic systems and on the ability of operating systems to address both small-scale, high-performance embedded systems as well as large-scale complex systems requiring higher-level OS constructs like exception handling, atomic computations, etc. Our work is best summarized as 'research addressing the configurability and adaptability of operating system kernels'. Our initial work addressed the process and then object levels (the GEM and CHAOS systems), whereas our current work is addressing configurability and adaptability at the threads level. Some of our results are reviewed below, in the context of our operating systems constructed in the U.S.

4.2 Ada-supporting Runtime Systems

A continuing, DOD-induced thrust in current research on real-time systems is to design and build run-time support for real-time Ada. In [BP91], the authors present a brief description of some of the Ada 9X proposals that are intended to address hard real-time requirements. Ada 9X is a revision to the Ada programming language standard[Wel92]. The reports [Inc92, QD92, Sof92] present some results of real-time implementation studies of Ada. In [BJ87, Bak87, BJ86],

the authors present the interface of a run-time environment for real-time Ada. In [Bri92], the author deals with different possible time representations and their utilization in real-time Ada systems.

Corset and Lace[BJ87, Bak87, BJ86] are runtime environment interfaces. Corset is an interface specification for a compact runtime support environment for tasking for Ada. Lace is an interface specification for a low level adaptable common executive that implements a model of real-time, lightweight tasks. Compiled Ada tasks and programs request Corset and Lace services via normal Ada procedure calls.

Corset hides details of the runtime support environment (RSE) from the compiler. Lace, in turn, hides the details of processor allocation from the Ada RSE. This permits tailoring the dispatching policy to fit the application. In addition to information hiding, Lace also supports multiprogramming of simple Ada procedures without involvement of the Ada RSE, thus, eliminating unnecessary inefficiency and unpredictability. Such multiprogrammed procedures can be executed alongside other tasks that make use of the full Ada RSE. Therefore, it is easy to construct hybrid systems. Execution timing remains under control of the Lace dispatcher. Lace does not provide directly for intertask communication or memory management. Such services are provided separately, possibly using the Lace operations. In [BJ87], the authors discuss the Corset interface and the Lace interface in detail.

4.3 The Predictable Execution of Real-time Programs

The general characteristics of a real-time kernel include[SR87] its size may be adjusted to each application's needs, multitasking with low overhead for task context switch, quick response to external interrupts, no or limited use of virtual memory, support for time-constraints in task such as priority scheduling, support for real-time clocks, special alarms and time outs, and primitives to delay, pause, and resume tasks. This section presents the design a few well-known real-time kernels, excluding our past work on the GEM system[SBWT87].

4.3.1 Spring Kernel

The goals of Spring project at the University of Massachusetts[SR87] are: the development of dynamic, distributed, online real-time scheduling algorithms, the development and implementation of Spring kernel which supports a network of multiprocessors, the development of multiprocessor nodes in order to directly support the kernel and the scheduling algorithm, and the development of real-time tools.

The Spring system is physically composed of a network of multiprocessors, but only one multiprocessor node was ever built. Each multiprocessor contains at least one application processor, one or more system processors, and an I/O subsystem. Each processor has its own local memory which collectively form a global memory space.

Tasks, which can be periodic or non-periodic, are execution traces through programs, and are the dispatchable and guarantee-able entities in the system. Non-periodic tasks have deadlines by which they must finish. Periodic tasks have recurring initializations and deadlines until they are terminated. All system tasks are resident in memory of the system processors. Although system tasks run on system processors, application tasks can run on both application and system processors by explicitly reserving time on the system processors. The Spring kernel contains task management primitives that utilize the notion of preallocation where possible to improve speed and to eliminate unpredictable delays.

The I/O subsystem is a separate entity from the Spring kernel and it handles non-critical I/O, slow I/O devices, and fast sensors. The I/O subsystem can be controlled by some other real-time kernel.

The design of the kernel is based on the principle of segmentation as applied to hard real time systems[SS87]. Segmentation is the process of dividing resources of the systems into units where the size of the unit is based on various criteria particular to the resource under consideration and to the application requirements. The goals of using segmentation in hard real-time systems are to develop well defined units of each resource, to increase understandability, and to allow to put these units together by an on-line algorithm in such a manner as to provide predictability with respect to timing constraints[SR87].

Scheduling: The system processors run the scheduling algorithms. The scheduling algorithm separates policy from mechanism and is composed of four modules. At the lowest level there exists a dispatcher. The dispatcher is the mechanism of the scheduler and it simply removes the next task from a system task table (STT) which contains all guaranteed tasks already arranged in the proper order for the multiple application processors. The second module is a local scheduler. The local scheduler is responsible for locally guaranteeing that a new task can make its deadline, and for ordering the tasks properly in the STT. The third module is the global scheduler that attempts to find a site for execution for any task that cannot be locally guaranteed. The final module is a Meta Level Controller which has the responsibility of adapting various parameters by noticing significant changes in the environment and serving as

the user interface. In [RS84, SRC85, ZRS87a], the authors present and analyze the details of the scheduling algorithms. However, implementation data is not available at this time.

Memory Management: In the Spring kernel, the OS is core-resident. To eliminate large and unpredictable delays due to dynamic memory allocation (page faults and page replacements), the Spring kernel pre-allocates as much memory (for task control blocks, stacks, buffers etc.) as possible.

Inter-Process Communication: The kernel supports synchronization and communication with five IPC primitives: SEND, RECV, SENDW (send and wait), RECVW (receive and wait), CREATMB (create mailbox). Mailboxes are memory objects. The Spring kernel avoids the need for semaphores by implementing mutual exclusion directly in the schedule. This results in somewhat inflexible process scheduling because prematurely finishing processes cannot relinquish their extra execution time to other processes[SZ92].

4.3.2 Maruti

The main focus of the Maruti project[GMAT90, Agr90, AL87, MA90, YA89] at the University of Maryland is to examine the constructs of future distributed, hard real-time, fault tolerant, secure operating systems. Maruti is an object-oriented system whose basic building block is an object. Objects consist of two main parts: a control part (or joint) which is an auxiliary data structure associated with every object and a set of service access points (SAPs) which are entry points for the services offered by an object. Each joint maintains the object's information (such as computation time, protection and security information) and requirements (such as service and resource requirements). Timing information, maintained in the object, is dynamic and includes temporal relations among objects. A calendar, a data structure ordered by time, contains the name of the services that will be executed and the timing information for each execution.

In Maruti, each application is described in terms of a computation graph , a rooted directed acyclic graph. The vertices represent services and the arcs depict timing and data precedence between two vertices [LA87]. Objects communicate with one another by *semantic links.* Such links perform range and type checking of the information. Objects that reside in different sites needs *agents* as representatives on remote sites.

Maruti is organized in three distinct levels: the kernel, the supervisor, and the application level. The kernel is the minimum set of servers needed at execution time and consists of a set

of core-resident objects that include: a Dispatcher (invoked upon the completion of a service or at the start of another service), a Loader (to load objects into memory), a Time Server (provides the knowledge of time to executing objects), a Communication Server (for sending and receiving messages), and a Resource Manipulator.

The supervisor objects prepare the computation to take place, making timely execution possible through the pre-allocation of resources. The supervisor level objects in Maruti are: Allocator (extracts the resource requirements from the joints of the objects of the computation graphs of the request, then allocates and verifies the allocation of the resources), Verifiers (verifies resource usage and reservation), Binder (responsible for connecting communication objects, as well as for verifying that the semantic relation is properly established), Login Server (the user interface to Maruti whose major task is command interpretation), and Name Server (bridges different name spaces, keeps track of status of machine, and keeps information about location of machine).

In [NP91], the authors have described the application of partial evaluation to programming languages for hard real-time systems, and have described a partial evaluator for Maruti. The initial implementation of Maruti on a network of SUN Unix workstations is now being replaced by one on DecStations running a native kernel. As with Spring, this implementation strategy is not likely to result in a widely used system.

4.3.3 YARTOS

YARTOS (Yet Another Real-Time Operating System) is an operating system kernel that supports the construction of efficient, predictable, real-time systems [JSP91, Jef89b, Jef89a, Jef92]. The programming model supported by YARTOS is an extension of Wirth's discipline of real time programming [Wir77]. It is a message passing system with a semantics of inter-process communication that specifies the real-time response that an operating system must provide to a message receiver. These semantics provide a framework both for expressing processor-time dependent computations and for reasoning about the real-time behavior of programs. In [Jef89b], the author has described the programming model in detail.

YARTOS supports two basic abstractions: tasks and resources. A task is an independent thread of control that is invoked at sporadic intervals. The invocation intervals and deadlines for a task are derived from constructs in the higher level programming model. During execution, a task accesses a number of resources. A resource is a software object that encapsulates shared data and exports a set of procedures for accessing and manipulating data. Like a monitor,

objects require mutually exclusive access to the data they encapsulate. A set of tasks is said to be feasible if all requests of all the tasks will complete execution before their deadlines and no shared resource is accessed simultaneously by more than one task.

The sequencing algorithm for tasks is a variation of the well-known earliest deadline first (EDF) scheduling algorithm; a preemptive priority driven scheduling algorithm with dynamic priority assignment[LL73]. The novel feature of the algorithm is the fact it dynamically manipulates the deadline of a task invocation to ensure that the task maintains exclusive access to whatever shared resource it might be accessing. This manipulation of deadlines ensures that there will exist no contention for shared resources at run-time. Hence, YARTOS need not provide any special locking facilities for shared resources. Since tasks execute to completion in YARTOS, all tasks are executed on a single run-time stack. This improves memory utilization and reduces context switching overhead [Bak90].

4.3.4 ARTS

ARTS[TML90, TM89, MT90, TK88, TNR90] is a distributed real time operating system developed in the ART (Advanced Real-time Technology) project at Carnegie Mellon University. The goal of the ARTS operating system is to provide users with a predictable, analyzable, and reliable distributed real-time computing environment so that a system designer can analyze the system at the design stage and predict whether the given real-time tasks having various types of system and task interactions can meet their timing requirements.

Objects in ARTS can be passive or active. An active object contains one or more user defined threads. In the active object, the designer of the object is responsible for providing concurrency control among co-executing operations. An object can be defined by using C or RTC++[ITM92].

Each operation of an object has an associated worst case execution time, called a "time fence" value and a time exception handling routine. When the operation is invoked from a real-time thread, the operation is executed if there is enough remaining computation time allocated to the calling thread to complete the operation. Otherwise, the invocation is aborted and an exception is raised.

Real-Time Threads: ARTS provides real-time threads to users. Each thread has an associated procedure name and a stack descriptor which specifies the size and address of the thread's stack. A real-time thread can be hard real-time or a soft real-time thread. A hard real-time

thread must complete its activities by its deadline time whereas a for a soft real-time thread deadline is less important. A real-time thread can be defined to be a periodic or an aperiodic thread based on the nature of the activities.

Synchronization: ARTS provides Lock and Unlock primitives to delimit critical regions. When a thread wants to Lock an already locked variable, it is enqueued on a priority queue of threads. Also, if the priority of the calling thread is higher than the priority of the thread which is in the critical region, the priority of the thread in the critical section is raised to that of the calling thread. When the thread leaves the critical region, its original priority is restored.

Scheduling: ARTS kernel implement an integrated Time-Driven Scheduler(ITDS). The ITDS scheduler provides an interface between the scheduling policies and the rest of the operating system. An object oriented approach is used to implement the scheduler with the policies embedded in the scheduler object. Each instantiation of the scheduler may have a different scheduling policy governing the behavior of the object, with only one instantiation being active at a given time. These notions are similar to the ones developed earlier in the CHAOS operating systems.

Communication Scheduling: In the ART project, an extended rate monotonic scheduling paradigm is used in the communication scheduling[TMIM89] domain to allow the system to integrate the message scheduling and processor scheduling with a uniform priority management policy. In [Str], the author has developed an algorithmic scheduling model for the IEEE 802.5 Token Ring Network and proposed a modification to control algorithm of a token-ring adapter chip-set[MST89]. ARTS implements a communication structure which is intended to serve as a test-bed for new communication algorithms and protocols as well as new real-time hardware[TML90]. Protocols such as VMTP have been successfully implemented on the ARTS kernel. Furthermore, a Real-time Transfer Protocol (RTP) is developed to explore real-time communication issues. RTP features prioritized messages and a time fence mechanism. The RTP manager implements the RTP protocol and, thus, forms a single point through which remote communications must pass. Priority inversion is carefully avoided at this level. In [TTCM92], the authors have extended the Capacity-Based Session Reservation Protocol(CBSRP), which was proposed for realizing predictable real-time communications, to support dynamic control of Quality of Service (QOS). They have implemented and evaluated the extension of CBSRP on a Fiber Distributed Data Interface (FDDI) in ARTS.

The most interesting contributions of ARTS are their results regarding process monitoring and their recent research on multi-media communication protocols, using FDDI links, typically presented in the context of a the real-time Mach project, also being done at CMU. However, real-time Mach does not appear to be ready for distribution at this time. Other interesting research at CMU does not appear strictly connected to the ARTS project, such as the research on real-time transactions by Sha et al. and the work on rate-based scheduling by Lehocsky and his students.

4.3.5 HartOS

The HartOS real-time operating system is being constructed by Prof. Shin's group at the Univ. of Michigan, for a non-shared memory machine. The operating system appears to focus on support for on-line scheduling, in part targeting applications in autonomous robot control and multi-media applications. Its implementation is being performed on SUN 3 machines using the X-kernel communication software produced by Larry Peterson's group at the Univ. of Arizona[PHOA89]. While the OS appears rather standard in the primitives being offered, its novel attributes are its support for special communication protocols addressing multi-media applications and its rigorous study of on-line scheduling for distributed memory machines (sets of workstations), including machines used in manufacturing environments[HS91].

4.4 Complex Execution Environments

Operating systems addressing more complex distributed or parallel execution environments must offer a richer set of primitives than those offered by the systems reviewed above. This section reviews the Alpha and Chaos systems, and we comment on the facilities of TRON, as well.

4.4.1 Alpha

Alpha[JN90a, JN90b, NCS+90] is a non-proprietary operating system for large, complex, distributed real-time systems. Alpha arose from the Archons Project at Carnegie Mellon University, which offered a partially implemented prototype operational in 1987. Versions now run on Sun, Concurrent, and SGI hardware.

Alpha's kernel provides its clients with a coherent computer system which is composed in a reliable, network transparent fashion of an indeterminate number of physical nodes. Its principle abstractions are objects, operation invocations, threads. Objects are passive abstract data type

(code + data) in which there may be any number of concurrently executing activities. Each instance of a client level object has a private address space, and exists entirely on a single node. Objects can be dynamically migrated among nodes. Initial object placement is specified by the user. Objects may be transparently replicated, with members of the replicated set residing on different nodes. The kernel defines a suite of standard operations that are inherited by all client objects, and these standard operations can be overloaded. Objects are named by capabilities that are protected by the kernel and not directly accessible by applications. Alpha's kernel offers atomic transaction-controlled updates to an object's permanent representation.

Alpha threads are the unit of schedulability, and are fully preemptable. A thread is the locus of control point[NCS+90] movement among objects via operation invocation. It is a distributed computation which transparently and reliably spans physical nodes. A thread carries parameters and other attributes related to the nature, state, and service requirements of the computation it represents.

The invocation of an operation of an object is the vehicle for all interactions in the system, including operating system call. Threads move from object to object via invocation. Operation invocation has synchronous request/reply semantics.

Alpha's exception blocks are a kernel level mechanism for the specification of application specific consistency and correctness mechanism. If an exception occurs, control is returned to the appropriate exception handler where the operation can be retired or some other compensating action can be taken.

Alpha utilizes a transactional distributed computing model for trans-node concurrency control and integrity because it can be well integrated with Alpha's block structured management of real-time constraints and exceptions, and can be tailored to meet application-specific needs. Alpha's kernel provides transaction mechanism for atomicity, permanence, and application specific concurrency control individually.

The future of Alpha is unclear due to its funding from U.S. Navy sources, while DARPA has been pushing for real-time MACH instead.

4.4.2 *CHAOS*

CHAOS[1][SGB87, SB87, GGSW88, Sch88, Gop88, GS89a, GS89b, GS90, GS89c, GBSG89, SGZ90a, SGZ90b] is a a family of object-based real-time operating system kernels that address portability, extensibility, and customizability for low-level and subsystem-level operations. The family is *extensible* in that new abstractions and functionalities can be added easily and effi-

[1] A **C**oncurrent, **H**ierarchical, **A**daptable **O**perating **S**ystem supporting atomic, real-time computations.

ciently such that uniform kernel interfaces are maintained. This is useful because it permits the implementation of domain or target machine specific features while preserving some given kernel interface for existing programs. It also provides an environment for experimenting with and prototyping of new operating system constructs and policies.

The family is *customizable* in that existing kernel abstractions and functions can be modified easily. This is useful because it facilitates changes to an operating system for uses with different target architectures or application domains. The family is *portable* in that its implementation is based on the now widely accepted Mach cthreads standard[SFG+91] as a base layer for uniprocessors and parallel architectures – called the CHAOS^{base} member of the kernel family. However, upwardly compatible modifications have been made to the cthread interface in order to accommodate real-time applications[ZS91]. Extensibility and customizability of the family are attained by use of the object model for description of the operating system's interface[SGB87, HFC76] and for operating system and user program implementation. Each application program is composed of a number of user objects, which use system-defined objects to access operating system services. However, as opposed to other object-based operating system kernels[SGB87] and in order to attain extensibility and customizability, family members do not describe their system interfaces by exporting some system objects (i.e., their classes[HFC76]). Instead, the exported object classes are refined with two novel abstractions supported by CHAOS^{min} : *attributes* and object *flavors*. CHAOS^{min} is the lowest-level object-based member of the kernel family. *Attributes* are abstract properties that can be associated with classes, objects, object states, operations, and invocations. The *flavor* of an object is defined as its set of permissible attributes. CHAOS^{min} neither defines nor interprets attributes; it merely passes them to a special type of object supported by CHAOS^{min} – termed a *policy* object. Such policies may be associated with objects, thereby providing the implementation of the object's flavor.

Exactly one policy object is associated with each object (user or system). Such a policy has complete control over the object's execution, and can therefore define the object's flavor and interpret and enforce its attributes. Policy objects are invoked implicitly as a result of operations (e.g., creation, invocation, ...) on the objects they manage. Each policy is itself implemented as an object and may use a limited form of inheritance for implementation of new functionality.

Objects are extended and customized, then, by changing their flavors and attributes instead of changing their operations. As a result, the interfaces of user programs to system objects need not be altered when policies are changed to support additional attributes, when object flavors

are changed, or when the implementations of policies are varied. The resulting structure of the kernel family consists of three components: (1) the *nugget* implementing CHAOS^{base}, which is a real-time cthreads package[2][ZS91], (2) the *vanilla layer* implementing CHAOS^{min}, and (3) the *policies* or application implementing certain CHAOS^{arc} flavors and attributes. CHAOS^{base} is the machine dependent component that implements the basic abstractions used by the remainder of CHAOS^{min} : real-time *execution threads*, *virtual memory regions*, and *synchronization primitives.*

The *vanilla layer* is the fixed, machine independent component that implements the functionality of CHAOS^{min} : classes, objects and invocations. It supports the following built-in object flavors: *ADT*, *Monitor*, *TADT*, and *Task.* A primitive object of flavor *ADT* (abstract data type) is passive[SGB87] and has well-defined internal state. Its operations are executed in the address space and by the execution thread of the invoker (caller). An object of type *monitor* is a passive object that allows exactly one execution thread at a time to execute its operations. Monitor objects behave like Hoare monitors, with the exception that their explicitly specified scheduling policies (for the selection of invocations to be executed) may differ among instances. An object of flavor *TADT* (threaded abstract data type) is active and is used for the representation of parallelism in CHAOS^{arc} applications. A TADT object creates and starts a new execution thread for the execution of each operation invocation. A CHAOS^{min} *task* object is like an Ada task in that it consists of a single active thread of control and has multiple entry points selected by this thread. The vanilla layer does not implement any invocation attributes or special invocation semantics. All object operations (creation, deletion and invocation) go through the vanilla layer. If such operations involve the use of non built-in flavors or attributes, the vanilla layer redirects the operation to the appropriate policy operation using well-defined rules.

The CHAOS^{arc} family member. The most interesting set of policies constructed with CHAOS^{min} addresses the predictable execution of highly dynamic real-time programs. They are called as the CHAOS^{arc} kernel. CHAOS^{arc} permits the reliable execution of real-time software, where (1) computations must complete within well-defined timing constraints typically captured by execution *deadlines*, and (2) programs must exhibit predictable behavior in the presence of uncertain operating environments. (2) is achieved by provision of operating system constructs that may be used to *guarantee* desired performance and functionality levels of selected computations in real-time applications[GS89a] – termed *atomic,real-time computations.* These constructs implemented by the CHAOS^{arc} object-based operating system kernel provides con-

[2] All threads created by a single user process share the process' address space yet have their own execution states.

structs that deal with uncertainty by allowing programs to be *adaptable* (i.e., changeable at run-time) in performance and functionality to varying operating conditions. The adaptations specifically supported by CHAOS^{arc} constructs are those that may be implemented as reactions to external events – termed *reactive adaptations*, as opposed to adaptations that anticipate changes in the operating environment – termed *preventive adaptations*[BS88, SBWT87, SGB87, GS89c]. However, programming and monitoring system support is implemented for CHAOS^{arc} so that preventive adaptations may be performed as well.

On-line monitoring. Another issue addressed by the CHAOS researchers is the application-specific, on-line monitoring of running real-time programs. The purpose of such monitoring is to use monitor data to adapt running programs in performance and functionality to changing external execution environments. The ideas presented in [KSO90, KS91, OSS90] are now being integrated into the lowest layers of CHAOS, thereby permitting implementors of specific abstractions at the threads or object levels[Muk91].

The CHAOS system is portable to multiple platforms due to its use of real-time threads as a lower layer. While the systems are not intended for commercial use, offshoots are being used in commercial robotics applications[BG92].

4.4.3 CTRON

CTRON[OWK87, WOK+87, KOOH87, bDKS89] is a part of the TRON[1][Sak87c] platform for real-time operating systems. The general TRON project is designed for network nodes consisting of different kinds of computers. The goal behind the design of CTRON is (1) to provide a high level of performance functions that are common to diverse network nodes and (2) to achieve software portability.

To assure software portability, the operating system is subdivided into two functional sections. One section consists of functions that hide the processor architecture and provide common interfaces; these functions are not portable among nodes with different processor architectures. The other section offers portable functions that assume a common interface.

Network nodes can be classified into various groups based on the kind of services they provide. A few examples are: Switching nodes (for circuit switching or packet switching, etc.), Communication processing nodes (voice storage service, facsimile communication processing, etc.), information processing nodes (files and data bases, data processing service etc.), and workstation nodes (to provide user-friendly interface to the end users). These groups, normally, have different operating system requirements. Switching nodes accommodate a number of

[1]This project studies the operating system interfaces/requirements for real-time processing. This project consists of several sub-projects, including ITRON[Mon87] for industrial embedded systems, BTRON[Sak87b, KTKS87] for business workstations, TRON CHIP[Sak87a] for a microprocessor used in the ITRON and BTRON, etc.

network nodes and have to be capable of processing a multiplicity of nodes and information simultaneously. Whereas, normally, workstation nodes process less than a hundred operations, and information processing nodes perform a few hundred or may be a thousand or so operations. To accommodate such varied requirements, the operating system interface is divided into two classes: (1) interfaces that can be used for all applications and (2) interfaces that are used selectively for specific applications.

The CTRON kernel provides a virtual processor model for the purpose of hiding the processor architecture. This model is characterized by a set of objects that indicate the abstraction of processor functions. The objects supported by the CTRON kernel are: tasks, synchronization, exceptions, timers, memory, interrupts, and black box.

Tasks: Tasks are the parallel processing units of a program. With the goals of real-time performance and high degree of multitasking, the CTRON kernel implements a two-level scheduling model. The task model consists of schedule functions required by high-level programs and those required by ordinary programs. The CTRON kernel supports a cheaper task abstraction called pseudo-tasks to diminish overhead which is used to implement batch polling form of processing. Pseudo-tasks are used selectively for high performance polling of circuit equipment. To reduce overhead in task creation, CTRON defines a dormant state which is a pre-ready state with needed system resources except a CPU scheduling right. It also defines a cyclic task schedule from the dormant waiting states with fixed time periods to improve the real-time processing capability.

Synchronization/Communication: The CTRON kernel offers optional functions for synchronization and communication between tasks. CTRON provides event flags, semaphores, and message boxes for simple synchronization, mutual exclusion, and message communication respectively. The CTRON kernel provides a logical lock function with no queuing of serially reusable resources to improve performance and predictability. It also provides an Ada-like rendezvous function as an operating system interface for synchronous communication between tasks.

Exceptions: The CTRON kernel defines an exception as an asynchronous interrupt signal to a task and distinguishes it from an asynchronous interrupt signal to a real processor. Exception management functions include the registration of exception-processing handlers corresponding to tasks, management of exception masks, generation of asynchronous exception from software

etc.

Timers: The CTRON kernel provides two kinds of timers: a system timer (one per system), and private timers (one or more per task). They are useful for real-time communication protocol processing.

Memory: The CTRON kernel provides two memory models: A common memory model (that applies regardless of the processor architecture), and a selectable memory model (that recognizes a virtual memory architecture). The selectable model makes efficient use of virtual memory. The memory management interface hides all hardware architectures from the users.

Interrupts: Interrupt related operations include registering interrupt handlers and setting and releasing interrupt masks. A pseudo interrupt generation operation from software is also possible. With the help of a mapping table, physical interrupt (intended for software operations) are changed to logical interrupts to increase portability of interrupt processing handlers created by users and intrinsic to the kernel.

Black Box: Some objects are strongly dependent on processor architecture or system configuration. In such cases, it is difficult to prescribe a common-use model, instead, individual interfaces for individual systems are needed. However, this presents an obstacle to software portability. CTRON defines a black box model for this purpose. It defines only system call names in the black box model; it does not define input and output conditions, error conditions, or side effects. In [OWK87], the authors discuss this model in detail.

The CTRON operating system model is divided into two groups: a common model group and a selectable model group. The various functions in the later group are further classified into several subgroups, based on the necessity of application fields. The kernel interface is divided into four parts: group for the common model, group for the advanced real-time model, group for the advanced complex function model, and group for the advanced virtual memory model. It is possible to provide a combination of these groups (altogether there are six subsets available) for various service systems.

Brief Evaluation: While TRON and CTRON are superior to other, specific operating systems for real-time control offered in the U.S. due to the attempt to construct a portable platform for use in different applications on different target machines, there remain issues regarding per-

formance and flexibility due to the rigid definition of low-level CTRON functions (e.g., cyclic task scheduling, or the single RPC semantics part of CTRON). It would be more appropriate if interfaces were defined such that alternative mechanisms and policies are easily added to the systems being constructed on the CTRON basis.

4.4.4 Communication Networks

Another direction of recent research involves design and development of real-time communication protocols. A running continuous media applications, such as full motion video, can occupy significant bandwidth of the computer resource. Although some compression schemes such as JPEG[Wal91], MPEG[Gal91], and px64[Lio91] have been suggested to reduce data size, high quality video frames are usually too large for a conventional local area network[TTCM92]. For asynchronous applications like interactive video/teleconferencing, end-to-end delay has to be bounded and observable jitter should be avoided[TTCM92]. Because of these temporal and spatial constraints, continuous media communication requires special resource management[ATW+89]. In order to overcome such spatial and temporal constraints of continuous communication media, a few transport protocols such as ST-II (Stream Protocol II)[ea90], SRP (Session Reservation Protocol)[ATW+89], XTP (Express Transport Protocol)[CA90], VMTP (Versatile Message Transport Protocol)[Che87], and fast lightweight transport protocols have been proposed. These protocols can be divided into two classes: reservation and non-reservation based protocol[TTCM92]. The ST-II, and SRP protocols reserve system resources such as processor execution time, buffers, and network bandwidth before transmitting any data. A similar resource reservation model, a real-time channel, has been proposed for a wide area network environment[FV90]. Such reservation of resources requires significant operating system support. On the other hand, VMTP and XTP transfer data on a best-effort basis and without any resource reservation. However, they do not guarantee on the end-to-end delay or jitter bound for a session[TTCM92].

The best known research efforts in this area are performed by Ferrari at UC Berkeley. Their recent research has resulted in the construction of a real-time IP protocol, called RTIP[Zha91]. Many other results regarding real-time communication are available from that group. In [TTCM92], Tokuda et al. present a Capacity-Based Session Reservation protocol(CBSRP) in order to provide guaranteed end-to-end delivery of data through resource reservation in a local area network environment. CBSRP differs from ST-II and SRP in its capability of changing quality of service (QOS) parameters of a session dynamically.

5 Future Work

Future work in real-time operating systems must address the topic of a common basis for real-time computing. At this time, several such bases are being developed, including real-time Mach in academics and by the U.S. Department of Defense, real-time POSIX Unix, and many commercial systems, ad CTRON in Japan. Unfortunately, wide differences exist among these different systems and convergence to a single system appears to require more research. A joint system may come, however, from future and current work on object-oriented operating systems, which are currently being developed both at SUN Microsystems and by IMB in conjunction with Apple Computer. However, these commercial efforts are not yet addressing real-time computing and applications.

References

[Agr90] A. Agrawala. Systems engineering approach to time-driven systems. In *CompEuro 90*, 1990.

[AL87] A. Agrawala and S. Levi. Objects architecture for real-time, distributed, fault tolerant operating systems. In *IEEE Workshop on Real-Time Operating Systems*, July 1987.

[And90] T. E. Anderson. The perfomance of spin lock alternatives for shared-memory multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):6–16, January 1990.

[ATW+89] D. Anderson, S. Tzou, R. Wahbe, R. Govindan, and M. Andrews. Support for continuous media in the dash system. Technical report, University of California, Berkeley, October 1989.

[Bak87] T. Baker. A corset for ada. Technical Report 86-09-07, Department of Computer Science, University of Washington, 1987.

[Bak90] T. P. Baker. A stack-based resource allocation policy for real-time processes. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1990.

[BBH84] F. Baccelli, P. Boyer, and G. Hebuterne. Single server queues with impatient customers. *Advances in Applied Probability*, 16:887–905, 1984.

[bDKS89] TRON ASSOCIATION Supervised by Dr. K. Sakamura, editor. *Outline of CTRON*. Original Ctron Specification Series. Ohmsha, 1989.

[BG92] T. E. Bihari and P. Gopinath. Object-oriented real-time systems: Concepts and examples. *IEEE Computer*, 25(12):25–32, December 1992.

[BJ86] T. Baker and K. Jeffay. A lace for ada's corset. Technical Report 86-09-06, Department of Computer Science, University of Washington, 1986.

[BJ87] T. Baker and K. Jeffay. Corset and lace: Adapting ada runtime support to real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1987.

[BP91] T. Baker and O. Pazy. Real-time features for ada 9x. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1991.

[Bri92] L. Briand. Time management for ada real-time systems. *Ada Letters*, 12(5), September 1992.

[BS88] T. Bihari and K. Schwan. A comparison of four adaptation algorithms for increasing the reliability of real-time software. In *Ninth Real-Time Systems Symposium, Huntsville, AL*, pages 232–241. IEEE, Dec. 1988.

[BS91a] T. Bihari and K. Schwan. Dynamic adaptation of real-time software. *ACM Transactions on Computer Systems*, 9(2):143–174, May 1991. Older version available from the Department of Computer and Information Science, The Ohio State University, OSU-CISRC-5/88-TR, newer version available from College of Computing, Georgia Institute of Technology, Atlanta GA, GTRC-TR-90/67.

[BS91b] Ben Blake and Karsten Schwan. Experimental evaluation of a real-time scheduler for a multiprocessor system. *IEEE Transactions on Software Engineering*, 17(1):34–44, Jan. 1991.

[BSS88] John P. Lehoczky B. Sprunt and Lui Sha. Expoiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In *Proceedings of Real-Time Systems Symposium, Huntsville, AL*, pages 251–258. IEEE, 1988.

[CA90] Protocol Engines Inc. CA. Xtp protocol definitions, revision 3.5. Technical Report PEI-90-120, September 1990.

[Che87] D. R. Cheriton. Vmtp: Versatile message transaction protocol. Technical report, Computer Science Department, Stanford University, January 1987.

[CL86] H. Y. Chang and M. Livny. Scheduling under deadline costraints: A comparison of sender-initiated and receiver-initiated approaches. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1986.

[CL90] Min-Ih Chen and Kwei-Jay Lin. Dynamic priority ceilings: A concurrency control protocol for real-time systems. *The Journal of Real-Time Systems*, 2:325–346, 1990.

[CL91] M. Chen and K. Lin. A priority ceiling protocol for multiple-instance resources. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1991.

[CSR88] Sheng-Chang Cheng, John A. Stankovic, and Krithi Ramamritham. Scheduling algorithms for hard real-time systems - a brief survey. In *Tutorial Hard Real-Time Systems*, pages 150–173. IEEE, 1988.

[DM89] Michael L. Dertouzos and Aloysius K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, December 1989.

[ea90] Topolcic et al. Experimental internet stream protocol, version 2 (st-ii). 1990.

[FV90] D. Ferrari and D. C. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communication*, 8(3), April 1990.

[Gal91] D. L. Gall. Mpeg: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4), April 1991.

[GBSG89] Prabha Gopinath, Tom Bihari, Karsten Schwan, and Ahmed Gheith. Operating system constructs for managing real-time software complexity. In *Proceedings of 1989 Workshop on Operating Systems for Mission Critical Computing, ONR, Maryland*, pages U1–U9. ONR et al, Sept. 1989. Also available as Philips Technical Note TN-89-110 and published by IOS Press, Netherlands, as 'Mission Critical Operating Systems, Studies in Computer and Communication Systems, Vol.1'.

[GGSW88] Ahmed Gheith, Prabha Gopinath, Karsten Schwan, and Peter Wiley. Chaos and chaos-art: Extensions to an object-based kernel. In *IEEE Computer Society Fifth Workshop on Real-Time Operating Systems, Washington, D.C.* IEEE, April 1988.

[GJ77] M. R. Garey and D. S. Johnson. Two-processor scheduling with start-times and deadlines. *SIAM Journal of Computing*, 6(3), 1977.

[GK68] B. V. Gnedeko and L. N. Kovalenko. Elements of queueing theory. In *Israel Program for Scientific Research*, 1968.

[GMAT90] O. Gudmundsson, D. Mosse, A. Agrawala, and S. Tripathi. Maruti: A hard real-time operating system. In *Second IEEE Workshop on Experimantal Distributed Systems*, pages 29–34, October 1990.

[Gop88] Prabha Gopinath. *Programming and Execution of Object-Based, Parallel, Hard Real-Time Applications.* PhD thesis, Department of Computer and Information Sciences, The Ohio State University, June 1988.

[GS89a] Ahmed Gheith and Karsten Schwan. Chaos-art: Kernel support for atomic transactions in real-time applications. In *Nineteenth International Symposium on Fault-Tolerant Computing, Chicago, ILL*, pages 462–469, June 1989. Also see GIT-ICS-90/06, College of Computing, Georgia Tech, Atlanta, GA 30332.

[GS89b] Ahmed Gheith and Karsten Schwan. Chaosart: A predictable real-time kernel. In *Butterfly Users Group Meeting, BBN Advanced Computers INc., Rochester, NY*, April 1989. Talk abstracts do not appear in proceedings.

[GS89c] Prabha Gopinath and Karsten Schwan. Chaos: Why one cannot have only an operating system for real-time applications. *SIGOPS Notices*, pages 106–125, July 1989. Also available as Philips Technical Note TN-89-006.

[GS90] Ahmed Gheith and Karsten Schwan. Chaos-arc – kernel support for multi-weight objects, invocations, and atomicity in real-time applications. Technical report, GIT-ICS-90/06, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, Jan. 1990. To appear in ACM TOCS.

[HFC76] A. Habermann, L. Flon, and L. Cooprider. Modularization and hierarchy in a family of operating systems. *Communications of the ACM*, 19:266–72, 1976.

[HKL91] M. Harbour, M. Klein, and J. Lehoczky. Fixed priority scheduling of periodic tasks with varying execution priority. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1991.

[HS91] C. Hou and K. Shin. Load sharing with consideration of future task arrivals in heterogenous distributed real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1991.

[HS92] Chao-Ju Hou and Kang G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. In *To be Published in Proceedings of IEEE Real-Time Systems Symposium*, 1992.

[Inc92] Tartan Inc. Evaluation of the implementation of multi-way select and the asynchronous transfer of control constructs. Technical Report LSN-039-UI, Ada 9X Language Study Note, March 1992.

[ITM92] Y. Ishikawa, H. Tokuda, and C. Mercer. An object-oriented real-time programming language. *IEEE Computer*, 25(10):66–73, October 1992.

[Jef89a] K. Jeffay. Analysis of a synchronization and scheduling discipline for real-time tasks with preemption constraints. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 295–305, December 1989.

[Jef89b] K. Jeffay. *The Real-Time Producer/Consumer Paradigm: Towards Verifiable Real-Time Computations.* PhD thesis, University of Washington, Department of Computer Science, 1989.

[Jef92] Kevin Jeffay. Scheduling sporadic tasks with shared resources in hard-real-time systems. In *To be Published in Proceedings of IEEE Real-Time Systems Symposium*, 1992. Also as Technical Report TR90-038, Department of Computer Science,University of North Carolina at Chapel Hill.

[JLT85] E. D. Jansen, C. D. Locke, and H. Tokuda. A time-driven scheduling model for real-time operating systems. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1985.

[JN90a] E. Jensen and J. Northcutt. Alpha: A non-proprietary os for large, complex, distributed real-time systems. In *Second IEEE Workshop on Experimental Distributed Systems*, October 1990.

[JN90b] E. Jensen and J. Northcutt. Alpha: An open operating system for mission-critical real-time distributed systems - an overview. In *Proceedings of the 1989 Workshop on Operating Systems for Mission-Critical Computing*, 1990.

[JSP91] K. Jeffay, D. Stone, and D. Poirier. Yartos: Kernel support for efficient, predictable real-time systems. In *Eighth IEEE Workshop on Real-Time Operating Systems and Software*, pages 8–12, May 1991.

[KM92] Tei-Wei Kuo and Aloysius K. Mok. Application semantics and concurrency control of real-time data-intensive applications. In *To be Published in Proceedings of IEEE Real-Time Systems Symposium*, 1992.

[KOOH87] I. Kogiku, T. Ohrui, T. Ohkuba, and Y. Hamada. A real-time portable operating system common to switching and information processing applications. In *Proc. International Switching Symposium*, pages 308–312, March 1987.

[KS91] Carol Kilpatrick and Karsten Schwan. Chaosmon – application-specific monitoring and display of performance information for parallel and distributed systems. In *ACM Workshop on Parallel and Distributed Debugging*, pages 57–67. ACM SIGPLAN Notices, Vol. 26, No. 12, May 1991.

[KS92] G. Koren and D. Shasah. D-over: an optimal on-line scheduling algorithm for overloaded real-time systems. In *To be Published in Proceedings of IEEE Real-Time Systems Symposium*, 1992.

[KSC86] J. F. Kurose, S. Singh, and R. Chipalkatti. A study of quasi-dynamic load sharing in soft real-time distributed computer systems. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1986.

[KSO90] Carol Kilpatrick, Karsten Schwan, and David Ogle. Using languages for describing capture, analysis, and display of performance information for parallel and distributed applications. In *International Conference on Computer Languages '90, New Orleans*, pages 180–189. IEEE, March 1990.

[KTKS87] M. Kobayashi, S. Takenouchi, Y. Kushiki, and K. Sakamura. The software structure of extended nucleus based on btron specification. In *Proc. Fall Joint Computer Conference on Exploring Technology:Today and Tomorrow*, October 1987.

[LA87] S. Levi and A. Agrawala. Temporal relations and structures in real-time operating systems. Technical Report CS-TR-1954, Department of Computer Science, University of Maryland, 1987.

[Law81] E. Lawler. Scheduling periodically occuring tasks on multiprocessors. *Information Processing Letters*, 12(1):9–12, February 1981.

[Lei80] D. W. Leinbaugh. Guaranteed response time in a hard real-time environment. *IEEE Transactions on Software Engineering*, January 1980.

[Lio91] M. Liou. Overview of the px64 kbit/s video coding standard. *Communications of the ACM*, 34(4), April 1991.

[LL73] C. W. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[LLN87] J. Liu, K. Lin, and S. Natarajan. Scheduling real-time, periodic jobs using imprecise results. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 252–260, December 1987.

[LM80] J. Y. Leung and M. L. Merrill. A note on preemptive scheduling of periodic, real time tasks. *Information Processing Letters*, 11(3):115–118, November 1980.

[LY82] D. W. Leinbaugh and M. Yamini. Guaranteed response time in a distributed hard real-time environment. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1982.

[MA90] D. Mosse and A. Agrawala. On fault tolerance in real-time environments. Technical report, Department of Computer Science, University of Maryland, March 1990.

[Mar91] E. P. Markatos. Multiprocessor synchronization primitives with priorities. In *Eighth IEEE Workshop on Real-Time Operating Systems and Software*, pages 1–7, May 1991.

[MCS91] J. Mellor-Crummey and M. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 9, 1991.

[MD78] A. K. Mok and M. L. Dertouzos. Multiprocessor scheduling in a hard real-time environment. In *Proceeding of The Seventh Texas Conference on Computer Systems*, November 1978.

[Mok83] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real Time Environment.* PhD thesis, M.I.T., 1983.

[Mon87] H. Monden. Introduction to itron, the industry-oriented operating system. *IEEE Micro*, pages 53–65, April 1987.

[Moo68] J. M. Moore. Sequencing n jobs on one machine to minimize the number of late jobs. *Management Science*, 17(1), 1968.

[MST89] T. Marchok, J. Strosnider, and H. Tokuda. Token-ring adapter-chipset architectural considerations for real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1989.

[MSZ90] L. D. Molesky, C. Shen, and G. Zlokapa. Predictable synchronization mechanisms for multiprocessor real-time systems. *Journal of Real-Time Systems*, 3(2), 1990.

[MT90] C. Mercer and H. Tokuda. The arts real-time object model. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 2–10, 1990.

[MT92] Clifford W. Mercer and Hideyuki Tokuda. Preemptability in real-time operating systems. In *To be Published in Proceedings of IEEE Real-Time Systems Symposium*, 1992.

[Muk91] Bodhisattwa Mukherjee. A portable and reconfigurable threads package. In *Proceedings of Sun User Group Technical Conference*, pages 101–112, June 1991. Techreport no: GIT-ICS-91/02.

[NCS+90] J. Northcutt, R. Clark, S. Shipman, D. Maynard, E. Jensen, F. Reynolds, and B. Dasarathy. Threads: A programming construct for reliable real-time distributed programming. In *Proc. International Conf. on Parallel and Distributed Computing and Systems*, October 1990.

[NP91] V. Nirkhe and W. Pugh. A partial evaluator for the maruti hard real-time system. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1991.

[OSS90] David M. Ogle, Karsten Schwan, and Richard Snodgrass. The dynamic monitoring of real-time distributed and parallel systems. Technical report, College of Computing, Georgia Institute of Technology, ICS-GIT-90/23, Atlanta, GA 30332, May 1990. To appear in IEEE TSE.

[OWK87] T. Ohkubo, T. Wasano, and I. Kogiku. Configuration of the ctron kernel. *IEEE MICRO*, april 1987.

[PHOA89] L. Peterson, N. Hutchinson, S. O'Malley, and M. Abbot. Rpc in the x-kernel. In *Twelfth ACM Symposium on Operating Systems*, pages 91–101. ACM, Dec. 1989.

[QD92] T. Quiggle and G. Dismukes. An analysis of the implementation and execution-time impact of ada 9x real-time features. Technical Report LSN-040-UI, Ada 9X Language Study Note, March 1992.

[RS84] K. Ramamritham and J.A. Stankovic. Dynamic task scheduling in hard real-time distributed systems. *IEEE Software*, 1(3):65–75, July 1984.

[RSL87] R. Rajkumar, L. Sha, and J. P. Lehoczky. On countering the effects of cycle-stealing in a hard real-time environment. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 2–11, December 1987.

[RSL88] R. Rajkumar, L. Sha, and J. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 159–169, 1988.

[RSL89] R. Rajkumar, L. Sha, and J. Lehocsky. An experimental investigation of synchronization protocols. In *Proceedings of 6th IEEE Workshop on Real-time Operating Systems and Software*, pages 11–17, May 1989.

[Sak87a] K. Sakamura. Architecture of the tron vlsi cpu. *IEEE Micro*, pages 17–31, April 1987.

[Sak87b] K. Sakamura. Btron: The business-oriented operating system. *IEEE Micro*, pages 53–65, April 1987.

[Sak87c] K. Sakamura. The tron project. *IEEE MIcro*, pages 8–14, April 1987.

[SB87] Karsten Schwan and Ben Blake. Experimental evaluation of a real-time scheduler for a multiprocessor system. Technical report, Computer and Information Science, The Ohio State University, OSU-CISRC-5/87-TR16, Sept. 1987. Also in IEEE TSE, Jan. 1991.

[SBWT87] Karsten Schwan, Tom Bihari, Bruce W. Weide, and Gregor Taulbee. High-performance operating system primitives for robotics and real-time control systems. *ACM Transactions on Computer Systems*, 5(3):189–231, Aug. 1987.

[Sch88] Karsten Schwan. Developing high-performance, parallel software for real-time applications. *Information and Software Technology, Butterworths Scientific Limited*, pages 218–227, May 1988.

[SFG+91] Karsten Schwan, Harold Forbes, Ahmed Gheith, Bodhisattwa Mukherjee, and Yiannis Samiotakis. A cthread library for multiprocessors. Technical Report GIT-ICS-91/02, College of Computing, Georgia Institute of Technology, 1991.

[SGB87] Karsten Schwan, Prabha Gopinath, and Win Bo. Chaos – kernel support for objects in the real-time domain. *IEEE Transactions on Computers*, C-36(8):904–916, July 1987.

[SGZ90a] K. Schwan, A. Gheith, and H. Zhou. Chaos-arc: A kernel for predictable programs in dynamic real-time systems. In *Seventh IEEE Workshop on Real-Time Operating Systems and Software, Univ. of Virginia, Charlottesville*, pages 11–19, May 1990.

[SGZ90b] Karsten Schwan, Ahmed Gheith, and Hongyi Zhou. From chaos-min to chaos-arc: A family of real-time multiprocessor kernels. In *Proceedings of the Real-Time Systems Symposium, Orlando, Florida*, pages 82–92. IEEE, Dec. 1990.

[SK91] D. B. Stewart and P. K. Khosla. Real-time scheduling of sensor-based control systems. In *Eighth IEEE Workshop on Real-Time Operating Systems and Software*, May 1991.

[SL92] Wei-Kuan Shih and Jane W. S. Liu. On-line scheduling of imprecise computations to minimize error. In *To be Published in Proceedings of IEEE Real-Time Systems Symposium*, 1992.

[SLR86] L. Sha, J. P. Lehoczky, and R. Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling. In *Proceedings of IEEE Real-Time Systems Symposium*, 1986.

[Sof92] RR Software. Runtime implementation strategies for protected records, multiway entry cal, and asynchronous transfer of control. Technical Report LSN-041-UI, Ada 9X Language Study Note, March 1992.

[SR87] J. Stankovic and K. Ramamritham. The design of the spring kernel. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 146–157, December 1987.

[SRC85] J. Stankovic, K. Ramamritham, and S. Cheng. Evaluation of a bidding algorithm for hard real-time distributed systems. *IEEE Transactions on Computers*, C-34(12), December 1985.

[SRL90] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[SS87] J. Stankovic and L. Sha. The principle of segmentation. Technical report, Department of Computer Science, University of Massachusetts, 1987.

[SSL89] Brinkley Sprunt, Lui Sha, and John Lehoczky. Aperiodic task scheduling for hard-real-time systems. *The Journal of Real-Time Systems*, 1:27–60, 1989.

[Str] J. Strosnider. *Real-Time Communication.* PhD thesis, Department of ECE, Carnegie Mellon University.

[SZ92] Karsten Schwan and Hongyi Zhou. Dynamic scheduling of hard real-time tasks and real-time threads. *IEEE Transactions on Software Engineering*, 18(8):736–748, Aug. 1992.

[SZG91] Karsten Schwan, Hongyi Zhou, and Ahmed Gheith. Multiprocessor real-time threads. *Operating Systems Review*, 25(4):35–46, Oct. 1991. Also appears in the Jan. 1992 issue of Operating Systems Review.

[TK88] H. Tokuda and M. Kotera. A real-time tool set for the arts kernel. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1988.

[TM89] H. Tokuda and C. Mercer. Arts: A distributed real-time kernel. *Operating Systems Review*, 23(3):29–53, July 1989.

[TMIM89] H. Tokuda, C. Mercer, Y. Ishikawa, and T. Marchok. Priority inversions in real-time communication. In *Proceedings of IEEE Real-Time Systems Symposium*, December 1989.

[TML90] H. Tokuda, C. Mercer, and J. Lehoczky. Scheduling theory and practice in arts. Technical report, School of Computer Science, Carnegie Mellon University, August 1990.

[TNR90] H. Tokuda, T. Nakajima, and P. Rao. Real-time mach: Toward a predictable real-time system. In *Proceedings of USENIX Mach Workshop*, October 1990.

[TTCM92] H. Tokuda, Y. Tobe, S. Chou, and J. Moura. Continuous media communication with dynamic qos control using arts with an fddi network. Personal Communication, October 1992.

[Wal91] G. Wallace. The jpeg still picture compression standard. *Communications of the ACM*, 34(4), April 1991.

[WC87] C. M. Woodside and D. W. Craig. Local non-preemptive scheduling policies for hard real-time distributed systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 12–16, December 1987.

[Wel92] A. Wellings. Ada run-time environment working group meeting summary. *Ada Letters*, 12(5):30–35, September 1992.

[Wir77] N. Wirth. Toward a discipline of real-time programming. *Communications of the ACM*, 20(8):577–583, August 1977.

[WOK+87] T. Wasano, M. Ohminami, Y. Kobayashi, T. Ohkubo, and K. Sakamura. Design principles and configurations of ctron. In *Proceedings of Fall Joint Computer Conference on Exploring Technology: Today and Tomorrow*, pages 159–166, October 1987.

[YA89] X. Yuan and A. Agrawala. A decomposition approach to nonpreemptive real-time scheduling. Technical Report CS-TR-2345, Department of Computer Science, University of Maryland, November 1989.

[Zha91] L. Zhang. Virtual clock: A new traffic control algorithm for packet-switched networks. *ACM Transactions on Computer Systems*, 9(2):101–124, March 1991.

[ZRS87a] W. Zhao, K. Ramamritham, and J. Stankovic. Scheduling tasks with resource requirements in hard real-time systems. *IEEE Transactions on Software Engineering*, May 1987.

[ZRS87b] Wei Zhao, Krithi Ramamritham, and J. A. Stankovic. Preemptive scheduling under time and resource constraints. *IEEE Transactions on Computers*, C-36(8):949–960, August 1987.

[ZS91] Hongyi Zhou and Karsten Schwan. Dynamic scheduling for hard real-time systems: Toward real-time threads. In *Proceedings of Joint IEEE Workshop on Real-Time Operating Systems and Software and IFAC Workshop on Real-Time Programming, Atlanta, GA*. IEEE, May 1991.

[ZSA91] Hongyi Zhou, Karsten Schwan, and Ian Akyildiz. Performance effects of information sharing in a distributed multiprocessor real-time scheduler. Technical report, College of Computing, Georgia Tech, GIT-CC-91/40, Sept. 1991. Abbreviated version in 1992 IEEE Real-Time Systems Symposium, Phoenix.

[ZSG92] Hongyi Zhou, Karsten Schwan, and Ahmed Gheith. The dynamic synchronization of real-time threads for multiprocessor systems. In *Symposium on Experiences with Distributed and Multiprocessor Systems, Newport Beach*, pages 93–107. Usenix, ACM, March. 1992.